REMARKS/ARGUMENTS

This amendment responds to the Office Action of December 12, 2006. The specification has been amended above to clarify that the term "Java" is a trademark of Sun Microsystems, Inc. In view of the examiner's restriction requirement, applicant previously elected claims 1-36 for prosecution and confirmed such election by letter mailed December 4, 2006. In accordance with MPEP § 821.02, applicant has amended the claims above to withdraw the non-elected claims (claims 37-48) without traverse. Further amendments are described below. The Exhibits referenced below have been separately transmitted under a "Cover Letter and Accompanying Declaration" being filed concurrently herewith. Claims 1-36 remain pending in the application.

I. Nonart Rejections

In the Action, claims 1-36 are rejected for indefiniteness under 35 U.S.C. § 112, second paragraph. Concerning independent claims 1, 13, and 25 (and hence all the remaining pending claims, which each depend from a corresponding one of these independent claims), it is asserted that the limitation "the operation" has insufficient antecedent basis. Paragraph (c) of claims 1, 13, and 25 and paragraph (d) of claims 13 and 25 have been amended to remove this limitation thereby overcoming this particular ground of rejection.

In separate respect to claims 2-3, 8, 14-15, 20, 26-27, and 32, these claims are rejected under 35 U.S.C. §112, second paragraph on the grounds that each contains either the trademark "Java" or "Sun Microsystems." (Specifically, claims 2, 14, and 26 refer to a "Java virtual machine," claims 3, 15, and 27 refer to a "Java Native Method Interface," and claims 8, 20, and 32 refer to a "Sun Microsystems virtual machine".) The Action cites Ex parte Simpson, 218 U.S.P.Q. 1020 (Bd. App. 1982) for the proposition that when a trademark or tradename is used in a claim, the claim is thereby rendered indefinite under this section because only the "source of goods" and "not the goods themselves" is being identified.

Applicant is aware that Sun Microsystems, Inc. has registered the mark "Java." To dispel any possible confusion, applicant has revised the specification so as to note the proprietary rights claimed by Sun in the Java name. Applicant has further amended claims 3, 15, and 17 to remove mention of this term. Concerning the remaining claims identified in this rejection, claims 2, 8, 14,

and 20, in the context of these particular claims, use of the term "Java" is both valid and appropriate for reasons that will now be discussed.

As a preliminary note, it is not necessarily fatal to the definiteness of an application that a trademark or tradename is used. Rather, it is necessary that "each case" be "decided on its own facts." In re Metcalfe and Lowe, 161 U.S.P.Q.2nd 789, 792 (C.C.P.A. 1969). See also In re Gebauer-Fuelnegg et al., 50 USPQ 125 (C.C.P.A. 1941) where four claims referencing the trademark "pliolite" were held to be valid. In accord are Ex Parte Jerry Kitten (BPAI, heard Sept. 16, 1999, Appeal No. 19960513); and Ex Parte William Z. Goldstein (BPAI, heard June 7, 2005, Appeal No. 20050823). The last two "unpublished" decisions are viewable at http://des.uspto.gov/Foia/BPAIReadingRoom.jsp.

The term "Java virtual machine" refers to a virtual machine having particular features generally recognized by those of ordinary skill in the art. Indeed, entire books have been written about the "Java virtual machine." Two are identified in McGuire et al. cited by the examiner (see U.S. 2003/0097360 A1 at par. 0002). Another such book is "Inside the Java Virtual Machine" by Venners, 2nd ed., McGraw-Hill, 1999, USA. Their existence, which predates the instant application, confirms that the term "Java virtual machine" had a recognized and particular meaning to those of ordinary skill in the art at the time the instant application was filed. This is unlike the situation with many trademarked products where the specifications of the product are not well defined in any literature.

In accordance with a shorthand way of defining the term "Java virtual machine," such term signifies a virtual machine that can interpret and execute Java bytecode, such as compiled from a Java application, and that can produce an output "customized" for a "particular platform," such as a host operating system. See Wikipedia, Java Virtual Machine, http://en.wikipedia.org/wiki/Java_virtual_machine (accessed February 2, 2007; Exhibit A).

Based on this shorthand definition, even if Sun Microsystems, Inc., the owner of the "Java" mark, were to turn around tomorrow and sell a completely unrelated product under the name "Java virtual machine," a machine meeting the above definition (and including the basic structure disclosed in Figures 1-3 of applicant's drawings) could be made, just as well, by those of ordinary skill in the art. This has already been demonstrated, for example, by the "Kaffe" implementation of a "Java virtual machine" which was made by a source independent of Sun

Microsystems and developed in a "clean room" environment (e.g., working backward from only the specified functions of a "Java virtual machine"). See Wikipedia, Kaffe, http://en.wikipedia.org/wiki/Kaffe (accessed February 4, 2007; exhibit B). Other implementations of "Java virtual machines" are available from yet other sources. Wikipedia, List of Java virtual machines, http://en.wikipedia.or/wiki/List_of_Java_virtual_machines (accessed February 4, 2007; exhibit C).

In short, the phrase "Java virtual machine" has an ordinary and customary meaning when examined "through the viewing glass" of a person skilled in the art. Home Diagnostics, Inc. v. Lifescan, Inc., 381 F.3d 1352, 1358, 72 USPQ2d 1276, 1279 (Fed. Cir. 2004). Thus the essential function of §112, second paragraph is fulfilled, which is to provide "adequate notice" to those of ordinary skill in the art of the "metes and bounds" of the invention. In re Goffe, 526 F2d. 1393, 1397, 188 USPQ 131, 135 (CCPA 1975) and In re Hammack, 427 F.2d 1378, 1382, 166 USPQ 204, 208 (CCPA 1970).

It is possible, indeed, to delineate the generally accepted boundaries of the term "Java virtual machine" as that term is understood by those of ordinary skill in the art. In its least restrictive sense, the term "Java virtual machine" is used to refer to a product that is able to interpret and execute Java bytecode (e.g., see exhibit A). In its most restrictive sense, the term refers to a product approved or certified by Sun Microsystems as passing the appropriate technology compatibility test. Provided a product falls into this fixed and settled range of meaning, it is appropriate to call that product a "Java virtual machine" (it will be noted, here, that the broader definition subsumes the narrower).

It will further be noted that even in its most restrictive and proprietary sense, the term "Java" in the phrase "Java virtual machine" functions more as a certification mark (i.e., attesting to the product meeting certain performance standards) than as a regular trademark (i.e., indicating a unique source). This certification or validation process is described, for example, in a FAQ entitled "Free and Open Source Java" posted at Sun's web site, www.sun.com/software/opensource/java/faq.jsp, as accessed February 2, 2007 (and reproduced, in part, in exhibit D).

Referring to the indicated portions of the above FAQ, in order to be certified by Sun Microsystems as being Java "compatible," a product must first pass an appropriate test using a

Technology Compatibility Kit (TCK) available from Sun. This test determines whether or not the product correctly implements a particular Java technology specification as defined by a "Java Specification Request" (JSR). The JSR, in turn, is created and updated (e.g., for later versions) in accordance with a "Java Community Process" (JCP) where final approval is made by a JCP Executive Committee (the mark "JAVA COMMUNITY PROCESS" was registered by Sun in 2001 under registration number 2501545). If an independent developer passes this test, they can then display the Java cup and steam logo (another Sun Microsystem's mark) to indicate that their product implementation has been officially recognized as being Java compatible. However, the developer must also agree to become a member of the "Sun Partner Advantage Program." Likewise, a developer can participate in the JCP by signing the "Java Specification Participation Agreement" or contribute to the "free and open-source projects" after signing the "Sun Contributor Agreement." This last agreement (and possibly the others) requires that the independent developer grant Sun "joint ownership in copyright and a patent license." This enables Sun "to serve the extended Java community" (including "traditional commercial" interests) and "to grow the Java ecosystem, for the benefit of all." In return, the independent developer is able to directly influence the future by "Taking Java where it hasn't been before" and to participate with peers "in an open community." This FAQ apparently followed Sun's announcement on November 13, 2006 entitled "Sun Opens Java" (www.sun.com/2006-1113/feature/story.jsp; accessed February 2, 2007; see exhibit E). The term "Java virtual machine," then, even taken in its most restrictive sense, signifies nothing more than that a developer elected to obtain Sun's validation of having produced a "compliant" implementation of such a machine.

In claims 2, 14, and 26 the term "Java virtual machine" is not only definite but also necessary for distinguishing the scope of these claims over the broader term "virtual machine" as used in claims 1, 13, and 25 (see exhibit F). Although conceivably a coined or substitute term might be used to draw this distinction, such as "Java-enabled virtual machine," such a term does not have an established meaning to those of ordinary skill in the art. Reference to a "Java virtual machine" would be needed, in any event, to afford the substitute term a definite meaning. As there is no adequate substitute that carries the equivalent meaning, the term "Java virtual machine" is "as precise as the subject matter permits." See Shatterproof Glass Corp. v. Libbey-Owens Ford Co., 758 F.2d 613, 624, 225 USPQ 634, 641 (Fed. Cir. 1985).

Claims 8, 20, and 32 refer to a "Sun Microsystems virtual machine." The term "Java HotSpot" is how Sun refers to their own implementation of the Java virtual machine (exhibit D). One might object that the reference to Sun Microsystems places the "metes and bounds" of the subject matter of these claims under Sun's proprietary control where it can be bent and twisted like a "nose of wax." However, even here, Sun is constrained by practical, if not legal, considerations from significantly alterating (as opposed to extending) the basic features of its own implementation. If Sun were to make .too large an alteration, its implementation would no longer be backward compatible with the legacy systems of its existing customers (thereby effectively breaching, for example, any license obligation to provide its existing base of customers with current updates).

Based on the foregoing, the rejection of claims 2-3, 8, 14-15, 20, 26-27, and 32, for indefiniteness under 35 U.S.C. §112, second paragraph, on the grounds that each contains the trademark "Java" or "Sun Microsystems," is overcome. In view of this and the further remarks below, these claims stand in condition for immediate allowance, which action is respectfully requested.

II. Art-based Rejections

A. Remarks pertaining to all pending claims

In the Action, the examiner rejected claims 1-36 for obviousness over McGuire et al. (US 2003/0097360 A1) in view of Li et al. (US 7,143,392 B2). In respect to the pending independent claims (claims 1, 13, and 25), it is asserted that McGuire shows a system for concurrent operation of plural computer applications comprising a shared object space storing at least one object accessible to each application and a processing unit connected to a display. It is further asserted that although McGuire doesn't show a monitor for collecting data on the operation of the shared object space to be processed by the processing unit into statistical data for graphical representation on the display (under subparagraph (c) of claims 1, 13, and 25), such a feature is shown by Li, and it would have been obvious to modify McGuire to add this feature. It is further asserted that although McGuire doesn't show such a monitor storing time varying data about the objects in the shared object space (under subparagraph (d) of claims 13 and 25), such a

feature is also shown by Li, and it would have been obvious to modify McGuire to add this feature as well. It is contended that sufficient motivation existed for such modifications

> "in view of the fact that monitoring the resources usage of the system can not only reduce the amount of logged data and logged scenarios where the causality relationship does not need to be fully captured but also provide an efficient management of time consumption by particular threads or applications consuming excessive time and resources within a shared space environment."

Notwithstanding the above, it would not have been obvious to modify McGuire by adding the monitoring system of Li. Furthermore, even if such modification were to have been made, the resulting combination still would not have anticipated the subject matter of each independent claim.

It is perhaps easier to recognize the differences between the cited art and the claimed subject matter by first considering their common ground. As in the instant application, McGuire recognized there are advantages to having multiple Java virtual machines share the same object space, which advantages included faster start up time (since, after initial loading into the shared space, the objects don't have to be reloaded for each new application) and more efficient usage of memory resources (since redundant objects are eliminated). (Compare the first full paragraph on page 14 of applicant's specification, describing the related disadvantages of the prior art, with paragraphs 0010-0011 of McGuire.) However, as McGuire points out, such object sharing raises the possibilities of "conflicts," that is, two or more threads running on different virtual machines may attempt to use the same object concurrently (e.g, so that an intermediate variable being used by one thread may be corrupted by the other). McGuire notes that such "conflicts" also occur between different threads in a single virtual machine, and details how this problem has traditionally been countered using "synchronization." Under this approach, the highest priority thread may acquire a "lock" on any synchronized code in an object, so as to reserve that code until that thread is done using it, whereupon the lock is "released" for use by the next highest priority thread. Alternatively, the highest priority thread may only give up the lock temporarily by issuing a "wait" signal, and then retake the lock when the next highest priority thread issues a "notify" signal (e.g., see, generally, paragraphs 0011-0015 of McGuire).

As McGuire pointed out, when this system of synchronization is applied to more than one application or virtual machine, difficulties may be encountered (see the last seven lines of

paragraph 0028 of McGuire). In particular, if the virtual machine of the thread holding the lock abruptly closes or crashes, the lock is never released for use by the threads belonging to the remaining applications or virtual machines. This, indeed, is one reason why generally the prior art has insisted upon keeping each virtual machine independent of the other (see applicant's specification, page 8, first four lines of the second full paragraph). For ease of reference, this problem will hereinafter be termed the "indefinitely suspended lock release" problem (see paragraph 0075 of McGuire). McGuire begins its departure from the teachings of the present invention in the way it handles this problem.

According to McGuire, to avoid the indefinitely suspended lock release problem, the scope of locking by a thread should be limited to the virtual machine of that thread. As McGuire puts it, this avoids "the scalability problems of having locks on a shared object that become effective globally (i.e., across multiple VMs)" (e.g., see paragraph 0031; also see paragraphs 0074-0075, 0082, 0097, 0100, and 0108). As another way of phrasing it (see paragraph 0030), McGuire speaks of monitors that only control "local" (versus global) access (each object has an associated monitor from which the highest priority thread acquires the lock for that object and by which the other waiting threads are put in a que). Because "by far the most common" situation is for a thread to enter the monitor and acquire the lock without "contention" (e.g., without conflict from another thread), much of McGuire is devoted to describing a "bi-modal" system using a "thin" (space-saving) monitor that inflates (when necessary because contention is present) to a "fat" monitor (e.g., capable of holding a que of threads) in a manner consistent with local control (e.g., see paragraphs 0017-0027, 0033-35, and 0077-0082, and 0108).

Turning now to applicant's teachings, the approach taken is markedly different. Referring to the pending independent claims (claims 1, 13, and 25), a monitor is claimed that is "associated with said shared object space" (e.g., that operates at the level of a "global" monitor contrary to McGuire's teachings). This monitor is capable of "collecting data" about the shared object space for sending to a processing unit, which processing unit processes this data "into statistical information" for "graphical" display. Although conventional synchronization methods are also used to control access to objects (page 15 of applicant's specification, top paragraph, lines 3-5) and although one application may possibly close or crash while its thread is still holding the lock (page 15, top paragraph, lines 7-9), no objects are removed from the shared space until all

waiting threads have used the object (page 15, top paragraph, lines 6-7) and the administrator at the display, preferably notified by an alarm, can use the data collected to take "corrective action" merely by releasing the lock (just as they would for a "deadlock alert;" see page 20, twelfth line from the bottom to page 21, second line from the top). Although this is an ex post facto approach, the code implementing the virtual machine is kept free of complex data structures relating to specialized monitoring procedures, thereby optimizing normal performance. Also, it should be borne in mind that the indefinitely suspended lock release problem should occur relatively infrequently. (Note that McGuire, though targeting this problem, admits it is "relatively unlikely;" see paragraph 0090.)

In the Action, it is conceded that McGuire does not show the claimed monitor structure. However, the Action asserts that the recited monitor is shown in Li and that one of ordinary skill would have been motivated to use the "monitoring component" of Li to monitor the shared object space of McGuire's system to "reduce the amount of logged data...where the causality relationship does not need to be fully captured" and to provide "efficient management of time consumption by particular threads or applications consuming excessive time and resources." This assertion, however, ignores several factors that would have militated against making this combination. First, McGuire already had an automated monitoring system in place (using thin/fat monitors that permited only local access), so that no need existed, under McGuire's teachings, for an alternative display-based monitoring system. Second, contrary to the proposed motivation, there is no logging system in McGuire that needed optimization. Third, again contrary to the proposed motivation, the types of locking problems and deadlocks that can lead to threads or applications "consuming excessive time" were already addressed by McGuire's automated monitoring system.

The fourth factor that would have weighed heavily against making the proposed combination is that such a combination would have rendered McGuire unsatisfactory for its intended purpose. In McGuire, one real concern was for reducing extraneous code wherever possible (it appears enough extra code and execution overhead were added by its specialized monitoring system to contemplate adding any more). Hence McGuire sought to preserve the existing "bi-modal" model using "thin" (whenever possible) and "fat" (wherever necessary) monitors and dismissed one approach that "would allow more space in an object for storing

locking information" with the caution that such space "is at a premium" (see paragraph 0081; also see paragraphs 0018-0019, 0022, 0077 and 0080). McGuire's teaching to keep its code as unencumbered as possible would have militated against combining McGuire's shared space system with a monitoring system of the type shown in Li.

The monitoring system of Li is designed for distributed component systems (in which the various methods or functions embodied in the software are treated as if they were discrete hardware components; refer, generally, to col. 5, lines 20-35 of Li). To use the Li monitoring system, the software embodying the virtual machine must first be "instrumented" (col. 9, lines 25-26). What this means is that extra "probing" code must be added to the software between whichever software components are of interest, which "probing code" is made transparent to each linked pair of component interfaces by adding (with even more code) a "stub" at one component interface and a "skeleton" at the other (refer to col. 3, line 58 to col. 4, line 12; also see col. 13, lines 9-21, and FIG. 3). One of ordinary skill sharing McGuire's desire for optimizing performance to whatever degree possible would have been disinclined to encumber McGuire's code with the extensive additions needed to implement Li's monitoring system. Li's system is more apt to a research or test bench than to commercial embodiments where performance is of paramount concern.

Even if one of ordinary skill were to have combined McGuire's shared space system with Li's monitoring system, the resulting combination still would not have anticipated the subject matter covered by claims 1, 13, and 25. Although Li mentions monitoring a "heap," it doesn't explicitly describe monitoring a "shared object space," and hence does not teach "a monitor associated with said shared object space" as recited by each of the independent claims (claims 1, 13, and 25).

Based on the foregoing, then, each of the pending independent claims, claims 1, 13, and 25, together with their dependent claims, claims 2-12, 14-24, and 26-36, respectively, define patentably over the art of record. Accordingly claims 1-36 stand in condition for immediate allowance. Moreover, as will now be shown, each of these independent claims, as amended, includes separate limitations that patentably distinguish that claim over the cited art. Each independent claim and its pertinent limitations will now be considered in turn.

B. Separate remarks pertaining to independent claim 1 and its dependent claims 2-12

Claim 1 has been amended to recite that the shared object space is not only "accessible to" but also "updateable by" the plural computer applications. This runs contrary to the conventional or accepted wisdom at the time of applicant's filing. Conventionally, each application must be given control only over the resources of its own virtual machine so that an object or other data type undergoing alteration in accordance with the process of one application can't be "corrupted" midstream by another application. Even McGuire, which allows plural applications "to access" the shared object space, bases its bi-modal, local locking system (which it contrasts with a "naïve" approach) on this shared object space being "read access" only (see paragraph 0083; also refer to paragraphs 0077-0082). As McGuire puts it, "if this were not the case, a first Java VM could write a value to a shared object, only to find later that it had been changed (unaccountably) by a second Java VM" (paragraph 0083). McGuire, then, at the time of applicant's filing, actually disparaged or taught away from the system of amended claim 1. This conventional thinking is even better understood when it is recognized that it is largely predicated on security concerns; that is, given that the Java language is designed for network portability, there was concern in the art that code arriving over a network from an untrustworthy source could intentionally corrupt the data of the receiving system.

Starting out from this conventional way of thinking, the present inventors struck out on a new and untested path when they explored the notion that it was not always "corrupting" for one application to change a data object still in process by another application. Closely examining this counterintuitive notion, they came to the realization that in several commercial environments it was actually desirable, in terms of performance, for one application to allow a data object to be, as it were, "corrupted" by another and then to continue processing that data object (see page 14, second full paragraph). Thus, in applicant's specification, the example is given of an online trading environment where the price for particular stock (as represented by an object) may reflect the participation of thousands of traders and "the faster a trading system reacts" (by allowing free updating of this object) "the more transactions an exchange can execute, increasing its commission while maximizing trader satisfaction" (see the paragraph beginning at the bottom of page 15). To draw a rough parallel to the thread locking context discussed above, the difference here is as large and significant as a thread issuing a "wait" instead of a "release" signal, insofar as

the principal application (like the thread) is only temporarily giving up object control rather than demanding full control throughout its entire execution cycle.

As phrased in the specific limitations of amended claim 1, while the prior art (represented by McGuire) may have taught a shared object space "accessible" (e.g., readable) by plural applications, it did not teach that this shared object space should be "updateable" by these same applications. It will be clear, from the above, that McGuire's teachings were firmly rooted in conventional thinking and did not reflect the advance here made by the present inventors. Of course Li's teachings, which do not relate in any way to a shared object space, add nothing on point, and the proposed combination of McGuire and Li would not have anticipated the subject matter of amended claim 1.

The foregoing discussion does leave open the question of security, which, after all, was the bugaboo that seemingly blocked the path eventually taken by the present inventors. In the trading example above, with data and code flying back and forth from numerous sources over a network, what if an untrustworthy source were to introduce code specifically designed to corrupt the objects in the shared object space? Applicant might have implemented a complex security protocol (perhaps, following McGuire's lead, relying on plural modes of operation) but the extensive code and execution overhead this would have required would have degraded the very performance they were trying to achieve. Instead, the present inventors conceived a much simplier display-based monitor system (had they looked to Li's implementation as a model, they might have dismissed this general approach out-of-hand as requiring too severe a degradation in performance). This display-based monitor system is recited in general manner by subparagraph (c) of claim 1. In this fashion, applicant exchanged the convenience of an automated system for greatly enhanced performance and flexibility (as further described below).

Based on the foregoing, then, amended claim 1 and its dependent claims, claims 2-12, patentably define over the art of record. Thus these claims stand in condition for immediate allowance, which action is respectfully requested.

C. Separate remarks pertaining to independent claim 13 and its dependent claims 14-24

Turning now to independent claim 13, this claim recites additional features of the monitor. In particular, subpararaph (d) of this claim refers to a "shared monitor space" storing

references to objects having time varying data pertaining to the shared object space, which time varying data may be "selectively sampled at a desired frequency." Contrary to the position taken in the Action, there is nothing in Li that shows monitoring of objects, in a shared space or otherwise, such that time varying object-oriented data is "selectively sampled at a desired frequency." In relation to such objects, Li at best suggests producing a display of "casualty" or parent/child relationships in which different objects would be represented as nodes that are linked in order of their execution upon test. Such a display, while presenting the objects, doesn't reference a "sample" of their "time varying" states (in a component model, the internal state of each component is considered irrelevant) nor does Li suggest repeating such sampling "at a desired frequency" (in Li, intercomponent events are logged at whatever moment is dictated by software execution). In accordance with applicant's teachings, the identified features of the monitor allow an administrator to flexibly take any "corrective action" that might be needed to recover from "serious problems" (whether caused by system instabilities or malicious tampering). See, generally, applicant's specification, page 20, the first full paragraph. This may include, for example, reverting the shared object space back to the state of a prior sample (each sampling, so to speak, represents a "snapshot" of the state of the objects that captures, as claimed, their "time varying data") from whatever interval in the recent past is desired (as provided for by performing such sampling "at a desired frequency"). See, generally, applicant's specification, page 21, the first full paragraph.

Based on the foregoing, then, amended claim 13 and its dependent claims, claims 14-24, patentably define over the art of record. Thus these claims stand in condition for immediate allowance, which action is respectfully requested.

D. Separate remarks pertaining to independent claim 25 and its dependent claims 26-36

Passing on to independent claim 25, this claim has been amended to better distinguish its subject matter over that of claim 13 and to recite certain features of the monitor system that further enhance its performance and flexibility. It will first be noted, by way of background, that the preamble of claim 25 refers to "plural computer applications" and that subparagraph (a) calls out a "shared object space" selectively connectable to each of these plural applications. Subparagraph (c) of claim 25, as amended, recites a monitor for this shared object space that is

capable of collecting data "including on respective objects registered for monitoring by each of said plural computer applications," which data is then processed for graphical display.

Support for the recited "registry" feature of amended claim 25 is found in applicant's specification beginning with the first full paragraph on page 21 to the first paragraph of page 23. One preferred approach, in particular, is described in the second paragraph on page 22, wherein "an application may register an object to be monitored," from among those in the shared object space, "such as by making a single call" (e.g., to a registry method) "with the object to be monitored as the only argument." Under such an approach, the code for registering a shared object for monitoring assumes the compact form of a reusable method (preferably labeled "registry") that can be invoked in the same manner as any standard method and preferably by, as described, "any application" (refer to the first four lines of page 22).

The finely granulated control and flexibility offered by the "registry" feature of amended claim 25 is suggested by the example given in the last paragraph on page 21 of applicant's specification. Invoking this registry feature, for example, a remote application on-the-fly, so to speak, can call for a monitor to be placed on any shared object of interest, such as one counting the number of matching trades in a particular stock. The most current data held in this now "registered" object is returned to the interested user of that remote application where it is available for continuous monitoring in the form of a "graphical representation" on the user's display. Such "application-defined" monitoring can likewise be invoked by an administrative application thereby flexibly extending the feature to cover the special case of "user-defined" monitoring by "an administrative user" or "group" (refer to page 20 of applicant's specification, middle and last paragraphs).

There is nothing in Li's monitoring system that would have even remotely suggested to one of ordinary skill the described "registry" feature of amended claim 25. Under Li's approach, at best every object represents a particular method or functional "component," and to achieve the desired goal of tracking a particular thread as it sequentially moves from its first to its last component, it is necessary to add software probes at any point in the object-defining classes where a method might be invoked. In other words, even leaving aside the fact that Li nowhere mentions monitoring a "shared object space," Li does not suggest targeting the monitoring "to respective objects registered for monitoring" (instead it teaches indiscriminately probing every

object) nor, in Li, is the choice of what gets "registered for monitoring" made "by each of ...plural computer applications" (instead, the choice is routinely made to monitor every object and the necessary code or probe "instruments" are added to a subject application in advance of its execution). Accordingly, Li fails to suggest a monitoring system having the particular features recited in amended claim 25.

Based on the foregoing, then, amended claim 25 and its dependent claims, claims 25-36, patentably define over the art of record. Thus these claims stand in condition for immediate allowance, which action is respectfully requested.

E. Separate remarks pertaining to dependent claims 3, 15, and 27

It is well to mention, in passing, that dependent claims 3, 15, and 27, as amended, are directed to subject matter that, once again, flew contrary to conventional wisdom at the time the application was filed. According to these claims, the virtual machine of each application is connected to the shared object space through a "Native Method Interface." In accordance with conventional wisdom, this circumvents the normal way of providing an application access to the shared resources (e.g., the shared object space) of a virtual machine, which is by loading code through a class loader. Referring to FIG. 2 of McGuire, the conventional way for an application to instantiate or create a new object for placement on the "heap" or object space is to call on the class loader subsystem 110. In one conventional version of this subsystem, a ranked hierarchy of class loaders search their respective class libraries or sources 120, 125, or 130 to determine if the particular class is present and, if that class is available from more than one source, the task of loading the class is delegated to whichever loader has access to the most trustworthy source. Loosely speaking, the "primordial" source with its "core" classes is regarded as the most trustworthy, followed by the "application" source with its "standard" classes, followed by the "extension" source with its user-defined classes, with any "network" source (not shown) considered the least trustworthy. The function of the "class loader cache" 180 is to keep straight which loaded classes came from which class loader or source so that untrustworthy code loaded by a dubious source, for example, is restricted from accessing, impersonating, or in some other manner "corrupting" other classes derived from more trustworthy sources. To do this, for each source, the class loader cache maintains a separate list naming the classes loaded by the

corresponding class loader (each list is called a "name-space"). This is why, for example, in FIG. 3 of McGuire, though the virtual machines 205 and 305 share a common heap 240, they have their own "class loader caches" 225 and 325 (and hence their own name-spaces).

Referring, now, to FIG. 5 of applicant's drawings, under dependent claims 3, 15, and 27, code from each application or virtual machine platform is loaded into the shared memory space 100 directly through a native access layer 102 without any vetting or tracking by a "class loader cache." This explains why there is only one "shared" name space in FIG. 5 (listing all the loaded classes without regard to source). Under conventional wisdom at the time the application was filed, although use of a native method interface might have been viewed as tolerable for the limited purpose of interacting with host resources only available in native code, using such an interface to provide the main gateway into the shared resources of the virtual machine for all code sources would have been seen as a flagrant security breach. In FIG. 5, for example, there would have been concern that a malicious application, say the C/C++ application 112, might try to substitute an untrustworthy type from its API library 104 for a "core" or critical type previously loaded from a trusted source, say the Java API library 106. Such security concerns, however, are adequately addressed in applicant's system by means of its display-based monitor system (e.g., item 401 of FIG. 10) as recited in each base claim of claims 3, 15, and 27. This monitor system can be used to quickly alert the administrative user of any "serious problem" so that such user can take any necessary "corrective action" (such as revoking an application's access privileges and reverting the system). In this manner, then, the systems of claims 3, 15, and 27 achieve compatibility with a wide range of native applications while retaining a streamlined, secure, and efficient code architecture well-suited to high-performance environments.

Based on the foregoing, then, dependent claims 3, 15, and 27, as amended, define patentable subject matter even apart from their respective base claims. Thus these claims stand in condition for immediate allowance, which action is respectfully requested.

III. Conclusion

As shown above, each of the independent claims currently pending in the present application (claims 1, 13, and 25) patentably define over the cited references, McGuire and Li, whether taken alone or in combination. It has further been shown how dependent claims 3, 15,

and 27 define subject matter separately patentable over the prior art. Accordingly, all the claims currently pending in this application, claims 1-36, stand in condition for immediate allowance, which action is respectfully requested.

If the examiner believes that a conference with applicant's undersigned representative would help advance this application to issue, he is invited to contact such representative at the telephone number provided below.

Applicant submits that no fees are required for entry of this Amendment. If it is determined that any fees are due, the Commissioner is hereby authorized to charge such fees to Deposit Account No. 03-1550.

Respectfully submitted

CHERNOFF, VILHAUER, MCCLUNG & STENZEL

Dated: February 21, 2007

By ______________________

Kevin L. Russell
Reg. No. 38,292
1600 ODS Tower
601 SW Second Avenue
Portland, OR 97204
Tel: (503) 227-5631

Certificate Of Mailing

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Mail Stop Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on February 21, 2007.

Dated: February 21, 2007

____
Kevin L. Russell